Django Hacking : Comment dresser les serpents guitaristes

Yohann GABORY — Bruno DUPUIS
Pilot Systems

11 juillet 2011

Plan

1. Introduction
2. Les Settings
3. L'Admin
4. Les Templates
5. Les autres éléments à ne pas oublier
6. Conclusion

Introduction

Django : framework Web basé sur python

Description

- Il reprend le principe du MVC
- Il implémente WSGI
- Il est disponible sous licence BSD

Historique

Histoire et origine

- Son nom est un hommage au célèbre guitariste de jazz manouche Django Reinhart
- Il est issu du monde du journalisme
- Développé depuis 2003 par Adrian Holovaty et Simon Willison
- Un approche tres pythonique et peu intrusive
- Il ne se met pas en travers du chemin du développeur
- On peut se servir du dynamisme et de la puissance de Python

Les Settings

Que sont les settings ?

Un ensemble de variables globales qui donnent des informations sur :

- La base de donnée
- L'adresse du site
- Le cache
- Etc...

Settings : La problématique

Django utilise des chemins absolus pour la localisation des medias, des templates etc...

- *Problème :* Ce n'est pas DRY
- *Problème :* Ça complique de beaucoup les changements d'environnement
  - Entre les différentes instances des développeurs
  - Entre l'instance de développement et la production
  - Chaque instance partage des informations communes qu'il faut dupliquer
- Ce mode de configuration « en dur » est parfois trop rigide.
- On aimerait pouvoir modifier ces configurations durant l'exécution et de manière persistente

Settings : Hacking

- Changer les chemins relatifs en chemins absolus
- Plusieurs fichiers de settings par instance
- Cas concrets chez Pilot Systems

Changer les chemins relatifs en chemins absolus

Le code magique

```
import os
project_path = os.path.dirname(os.path.abspath(__file__))
```

Plusieurs fichiers settings par instance

Exemples

- `prod_settings`
- `devel_settings`
- `local_settings`

Plusieurs fichiers settings par instance : exemple

settings.py

```
# *** settings.py ***
# <settings goes here>
from local_settings import *
```

local_settings.py

```
# *** local_settings.py ***
# <en production>
from prod_settings import *
# <en developpement>
from devel_settings import *
```

Jouer avec votre vcs

.hgignore à la rescousse

On gère via Gestionnaire de Version :

- `prod_settings`
- `devel_settings`
- `settings`

Seul `local_settings` est en dehors du gestionnaire de version.

SQLConfig

Problématique

- La configuration est en dur dans les settings.py
- Il y a des cas ou on souhaiterait pouvoir les modifier simplement

Solution

- Configuration en base de données
- Intégrer la solution dans Django
- Faire en sorte que ce soit le plus transparent possible
- Nous allons tirer parti de la souplesse de python

Show me the code

SQLConfig

```
class Config(models.Model):
  key = models.CharField(max_length=200, unique=True)
  datatype = models.CharField(max_length=200)
  value = models.CharField(max_length=200)

  def get_python_value(self):
    if self.datatype == "int":
      return int(self.value)
    elif self.datatype == "bool":
      return self.value in ("1", "t", "T")
    else:
      return self.value
```

Show me the code (once again)

SQLConfig

```
class SettingsProxy(object):
  def __init__(self, target):
    self.target = target

  def __getattr__(self, key):
    if key in self.target.ALLOWED_SQL_OVERRIDES:
      try:
        option = Config.objects.get(key__exact = key)
        return option.get_python_value()
      except:
        pass
    return getattr(self.target, key)

proxy = SettingsProxy(settings)
for d in gc.get_referrers(settings):
  if isinstance(d, dict) and "settings" in d:
    d["settings"] = proxy
```

L'Admin

L'interface d'administration Django

Ce que nous allons voir

- Qu'est ce que l'admin ?
- La problèmatique
- Hacking

Admin : qu'est ce que c'est ?

Des points forts évidents

- Une application permettant de manipuler les données de l'application
- Très configurable
- Auto-documentée
- Paramétrable via un fichier python par application (`admin.py`)

Admin : La Problématique

Il existe néanmoins des soucis bloquant

- L'`admin.py` ne fait pas tout
- Le code des templates est complexe
- Tous les hooks n'existent pas (par exemple modifier les `change_list_result.html`)

Admin : Hacking

Le plan d'attaque

- Comment surclasser les templates d'administration
- Ruser pour surclasser les templates qui, à l'origine ne peuvent pas l'être (exemple : `admin_list`)
- Les `SortedUnicode`
- Comment créer de nouvelles colonnes (`property`)

Admin : Surclassage des templates

Fonctionnement standard

Copier depuis le dossier des templates d'admin django le template qui nous intéresse dans :

<votre dossier de template>`/admin/application/`<le modèle à surclasser>/<le template d'admin>

Par exemple

Par exemple pour le modèle *categorie* de l'application *blog*, pour changer le template du formulaire d'édition du template, vous faites :

```
cp .../django/contrib/admin/templates/admin/change_form.html \
   mon_projet/templates/admin/app/model/admin/change_form.html
```

Admin : Surclassage des templates II

Parfois, ça ne suffit pas

Dans `change_list.html` on ne peut pas changer `change_list_result` :

```
{% block result_list %}
  {% if action_form and actions_on_top and cl.full_result_count %}
    {% admin_actions %}
  {% endif %}
  {% result_list cl %}
  {% if action_form and actions_on_bottom and cl.full_result_count %}
    {% admin_actions %}
  {% endif %}
{% endblock %}
```

Impossible n'est pas python

La solution

Créer un templatetags `result_list` qui va écraser le templatetag originel :

```
def result_list(cl):
    """
    Displays the headers and data list together
    """
    return {'cl': cl,
            'result_hidden_fields': list(result_hidden_fields(cl)),
            'result_headers': list(result_headers(cl)),
            'results': list(results(cl))}
template = "admin/options/change_list_results_custom.html"
result_list = register.inclusion_tag(template)(result_list)
```

Impossible n'est pas python : la suite

Dans les templates...

- Copiez le `change_list.html` de django dans votre projet
- Copiez le `change_list_results.html` de django dans votre projet
- Renommez le `change_list_results_custom.html`
- Hackez !

Admin : Sorted Unicode

La problématique

Dans l'interface d'admin, les models sont auto classés par ordre alphabétique

Les singes à trois bras

âme sensible s'abstenir

- une class `SortedUnicode` comme unicode avec un ordre de tri que l'on peut spécifier
- *monkey-patcher* la fonction `capfirst` pour qu'elle conserver l'ordre de tri défini dans `SortedUnicode`

Admin : Créer de nouvelles colonnes

Dans l'admin on ne veut pas toujours afficher des champs de models...

Les property à la rescousse : dans models.py

```
@property
def tag_names(self):
    """
    Return the tag names, as a list
    """
    return [ t['name'] for t in \
             tagging.models.Tag.objects.get_for_object(self).values('name') ]

def tags_admin(self):
    tags = ''
    for tag in self.tag_names:
        tags = tags+tag+', '
    return tags
```

et dans admin.py

```
list_display = ('tags_admin')
```

Custom Widgets

La problématique

Dans l'interface d'admin, on peut vouloir utiliser des contrôles plus intelligents

Custom Widgets

Hookir le rendu de l'admin

```
class MyModelAdmin(ContentAdmin):
  model = MyModel

  def formfield_for_dbfield(self, db_field, **kwargs):
    if db_field.name == 'my_field':
      truc = do_something()
      class MyCustomWidget(forms.TextInput):
        class Media:
          js = ("js/myCustomWidget.js",)

        def __init__(self,*args,**kw):
          super(PictureChooserWidget,self).__init__(*args,**kw)
          self.truc = truc

        def render(self, name, value, attrs=None):
          rendered ="""<input name="my_field" value="" class="vTextField"
                maxlength="40" type="text" id="id_my_field">%s"""%self.truc
          return mark_safe(rendered)
      kwargs['widget'] = MyCustomWidget
    return super(MyModelAdmin,self).formfield_for_dbfield(db_field,**kwargs)
```

Les Templates

Tour d'horizon des templates

Plan d'action

- Que sont les templates ?
- La problèmatique
- Templates : Hacking

Templates : définition

Définition

- Le T de MVC...
- Le modele de compilation d'un ensemble d'objet : le contexte
- Le HTML qui va accueillir les objets python

Templates : la problématique

Les soucis posés

- Les templatetags et les filter sont parfois trop pauvre
- Les Templates est ce vraiment DRY ?
- Le Javascript, c'est pas DRY

Templates : Hacking

Le DRY jusque dans les templates

- Les templatetags custom
- Les héritages de templates, les inclusions de templates
- Un python dans le café

les templatestags custom

Filter

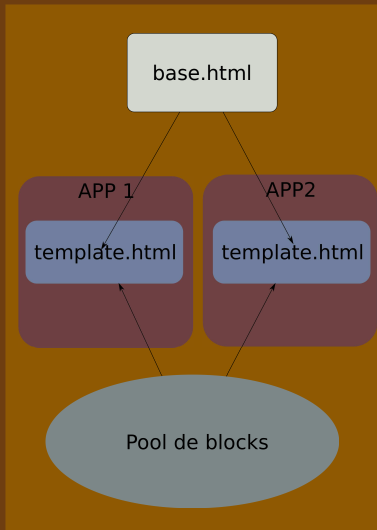
```
@register.filter(name='twitterparse')
@stringfilter
def twitterparse(value):
  p = ttp.Parser()
  result = p.parse(value)
  return mark_safe(result.html)
```

les templatestags custom

Tags

```
@register.inclusion_tag('blocks/explain.html')
def get_explanation(name):
  try:
    explain=Explanation.objects.get(name=name)
  except ObjectDoesNotExist:
    explain = None
  return {"explanation":explain}
```

Heritage et inclusion de template



Un serpent dans le café

Rendre javascript python-aware, un exemple avec float

```
var time_data_set = {
    all :
        {label:"all",
        data:{{graph_all}},
        points: { show: true },
        lines:{ show: true},
        color:0
        },
```

Un serpent dans le café II

Rendre javascript python-aware un peu plus loin

```
{% if hashtags %}
  var r = Raphael("holder",470,255);
  var r_10 = Raphael("hashtags_holder",470,255)
  var legends = new Array()
  var legends_10 = new Array()
  var hrefs = new Array()
  var all = new Array()
  var serie = new Array()
   {% for hashtag in hashtags %}
       {% ifchanged hashtag.name %}
          all.push(["{{hashtag.name}}",
                   {{hashtag.times}}])
       {% endifchanged %}
   {% endfor %}
{% endif %}
```

Les autres éléments à ne pas oublier

Middleware et Managements

Les MiddleWare

- `proccess_request`
- `proccess_view`
- `process_exception`
- `proccess_response`

Les Managements

- Comment ça marche ?
- Les managements custom
- Les Managements dans le vues (où ailleurs... d'ailleurs)

Les Middlewares

process_request

```
def process_request(self, request):
    """
    Here I can do whatever I want against the request
    """
    print request
```

process_view

```
def process_view(self,request,view_func,
                         view_args,view_kwarg):
    """
    Process the view to get info on the context
    """
    print view_func,view_args,view_kwarg
    return None #if I return something here it stops !
```

Les Middlewares, suite

process_exception

```
def process_exception(self,request,exception):
  print type(exception)
  print exception.args
  return None
```

process_response

```
def process_response(self,request,response):
  print connection.queries
  return response
```

Les Managements

Comment ça marche ?

```
python manage.py runserver
python manage.py syncdb
...
```

Les managements custom ?

```
def handle(self,user,requested_user=None,**options)
  user = DjangoUser.objects.get(username=user)
  fetch_tweets(user,requested_user=requested_user)
```

Lancer un management en python

```
from django.core.management import call_command
call_command('fetch', user,user_type)
```

Conclusion

Conclusion

Simplifier la vie des développeurs...

- ...par la souplesse de Python
- ...par la discretion de Django
- ... et pour ne garder que le plaisir !

Remerciements

Remerciements

- aux communautés Python et Django

Des questions ?